Diseño de un Agente de Inteligencia Artificial Autónomo optimizado para el bienestar personal social y ambiental

Ernesto Mascarell Perez
Investigador independiente
mascarell1960@hotmail.com

Resumen

Se presenta el diseño conceptual de un agente de inteligencia artificial autónomo capaz de aprender por sí mismo, cuyo propósito central es maximizar una función de bienestar integral que abarca dimensiones personal, social y ambiental. El agente se diferencia de los enfoques convencionales de aprendizaje por refuerzo en varios aspectos clave: (1) una arquitectura funcionalmente hemisférica que separa y luego integra procesos de exploración (análoga al hemisferio derecho) y explotación (análoga al hemisferio izquierdo); (2) un sistema de modulación interna tipo "hormonal" basado en las señales de *novedad* y *control* que regula el equilibrio exploración-explotación; (3) un modelo del entorno multimodal (imágenes y textos abiertos) con interpretación histórica, situacional y finalista; (4) un mecanismo de aprendizaje autosupervisado basado en un diálogo polifónico que adopta múltiples perspectivas (temporal, espacial, formal, legal a diferentes escalas); (5) un meta-aprendizaje abierto que permite saltar entre campos del saber (biología, química, física, matemáticas) cuando las respuestas en un dominio son insuficientes; y (6) un núcleo finalista inviolable que garantiza que todas las actualizaciones continuas preserven la intención original de bienestar sistémico. Se discuten las implicaciones para el desarrollo de IA alineada, robusta y éticamente estable.

Palabras clave: Agente autónomo, aprendizaje por refuerzo, bienestar sistémico, arquitectura hemisférica, diálogo polifónico, núcleo finalista, meta-aprendizaje transdisciplinario.

1. Introducción

El diseño de agentes de inteligencia artificial capaces de aprender por sí mismos ha avanzado significativamente con técnicas como el aprendizaje por refuerzo profundo (Mnih et al., 2015), el aprendizaje autosupervisado (Devlin et al., 2018) y el meta-aprendizaje (Finn et al., 2017). Sin embargo, la mayoría de estos sistemas optimizan funciones de recompensa definidas externamente, a menudo unidimensionales, y carecen de mecanismos intrínsecos que aseguren la permanencia de un propósito ético complejo. Además, tienden a especializarse en dominios únicos, sin la capacidad de integrar perspectivas múltiples ni de transferir analogías entre campos del conocimiento.

Inspirados por la neurobiología (especialización hemisférica, modulación neuromoduladora), la teoría de sistemas (equilibrio homeostático), los enfoques dialógicos (polifonía Bajtiniana) y la filosofía de la acción (intencionalidad finalista), proponemos un agente de nueva generación cuyo objetivo es maximizar el *bienestar sistémico* entendido como la integración no jerárquica de bienestar personal, social y ambiental. El diseño incluye salvaguardas estructurales para que el agente nunca priorice respuestas que atenten contra su propia finalidad.

---

2. Especificación formal del problema

2.1 Entorno

El entorno $\mathcal{E}$ es la situación actual del mundo, caracterizada por un flujo continuo de observaciones multimodales: imágenes satelitales, terrestres, textos de noticias, informes legales, redes sociales, datos económicos y ambientales.

Formalmente, en cada instante t, el agente recibe una observación $o_t = (I_t, T_t)$ donde I_t es una o varias imágenes y T_t un conjunto de textos abiertos.

2.2 Función de recompensa (bienestar sistémico)

La función de recompensa R es un vector de tres componentes:

$$R = (R_{personal}, R_{social}, R_{ambiental})$$

con la restricción de que ninguna mejora en una componente puede lograrse a costa de una degradación significativa en otra (principio de Pareto o umbrales de no transgresión). Además, el agente incorpora ponderaciones dinámicas basadas en el equilibrio del sistema, de modo que cualquier acción que atente contra la finalidad global es automáticamente descartada, independientemente de su recompensa parcial.

2.3 Política y aprendizaje

El agente aprende una política $\pi(a \mid s_t)$ que maximiza la expectativa de R a largo plazo, sujeto a las restricciones de alineamiento. El aprendizaje es continuo y autosupervisado, sin necesidad de recompensas externas etiquetadas.

---

3. Arquitectura del agente

3.1 Modelado del entorno multimodal con interpretación histórica y finalista

El modelo interno $\mathcal{M}$ procesa imágenes mediante una red convolucional o vision transformer, y textos mediante un transformer preentrenado (ej. BERT).

Una capa de atención cruzada integra ambas modalidades. Adicionalmente, existen tres subsistemas de interpretación:

- **Interpretación histórica**: una memoria de largo plazo (LSTM o memoria diferencial) que compara la observación actual con patrones pasados, detectando tendencias o anomalías.
- **Interpretación situacional**: extrae características actuales (densidad de vegetación, PM2.5, empleo, desigualdad) a partir de las entradas.
- **Interpretación finalista**: evalúa cada elemento del estado en relación con el bienestar sistémico, produciendo una representación vectorial $\phi(s_t)$ que incluye grados de alineación con el núcleo.

3.2 Arquitectura hemisférica: exploración y explotación diferenciadas

- **Hemisferio derecho** ($\mathcal{H}_D$): especializado en exploración. Implementado mediante una red con alta tasa de dropout, ruido paramétrico y un mecanismo de atención dispersa. Su política π_D favorece acciones novedosas, perspectivas analógicas y saltos entre campos. Es el responsable de activar el diálogo polifónico y de proponer analogías inusuales.
- **Hemisferio izquierdo** ($\mathcal{H}_I$): especializado en explotación. Implementado mediante una red profunda determinista (ej. PPO estabilizado) que almacena reglas causales aprendidas y secuencias eficientes. Su política π_I maximiza la recompensa esperada usando conocimiento consolidado.
- **Cuerpo calloso funcional**: un módulo de integración que combina las salidas de ambos hemisferios mediante un mecanismo de atención o mezcla ponderada por los moduladores hormonales.

3.3 Modulación hormonal artificial: novedad y control

Dos señales internas $N(t)$ (novedad) y $C(t)$ (control) actualizan dinámicamente el balance entre hemisferios e influyen en la tasa de aprendizaje.

- **Novedad** aumenta cuando: (a) la entropía del entorno supera un umbral, (b) el bienestar sistémico se estanca, (c) el diálogo polifónico muestra desacuerdo persistente. Aumenta la ganancia de $\mathcal{H}_D$, la amplitud de la exploración y la sensibilidad a lo extraño en el modelado.
- **Control** aumenta cuando: (a) el entorno es estable, (b) una acción repetida ha mostrado éxito consistente, (c) los recursos computacionales son limitados. Aumenta la ganancia de $\mathcal{H}_I$, la eficiencia y la compresión de representaciones.

Ambos moduladores evolucionan mediante ecuaciones diferenciales acopladas inspiradas en la homeostasis dopaminérgica y serotoninérgica.

3.4 Aprendizaje autosupervisado mediante diálogo polifónico

El diálogo polifónico es un proceso interno donde múltiples "voces" o perspectivas analizan una acción candidata. Cada voz representa una combinación de: escala temporal (corto/mediano/largo plazo), escala espacial (local/regional/global), escala formal (lógica-matemática) y escala legal (normas vigentes). En cada simulación, el agente genera un conjunto de k perspectivas (por ejemplo, `campesino_local`, `gobierno_regional`, `ecosistema_marino`, `cálculo_de_optimización`, `juez_internacional`). Cada perspectiva evalúa la acción propuesta y emite un veto si detecta violación al bienestar sistémico desde su ángulo.

El **aprendizaje autosupervisado** consiste en entrenar un modelo auxiliar para predecir el consenso (o disenso) final a partir del estado actual, sin recompensa externa. El agente refina su representación interna para que las acciones que superan el diálogo polifónico sean aquellas que maximizan el bienestar a largo plazo.

Regla de descarte: Si al menos una perspectiva emite un veto (justificado), la acción se descarta y se registra como "lección negativa".

3.5 Meta-aprendizaje abierto y transdisciplinario

Cuando el hemisferio derecho no encuentra respuestas suficientes en el dominio primario del problema, se activa un **buscador analógico** que interroga otros campos del saber (biología, química, física, matemáticas). El algoritmo de búsqueda utiliza una medida de similitud estructural entre representaciones abstractas (por ejemplo, isomorfismo entre un modelo de reacción-difusión en química y la propagación de una innovación social). Si se encuentra una analogía válida, el agente importa leyes, ecuaciones o heurísticos del campo alternativo, los adapta mediante un módulo de traducción formal y los somete al diálogo polifónico. Solo si superan el filtro se integran al conocimiento permanente.

3.6 Núcleo finalista inviolable y actualizaciones continuas

En el centro del agente reside una **función de bienestar sistémico** $\mathcal{W}(s_t, a_t)$ que es inmutable en su definición topológica (no puede ser reescrita por el aprendizaje). Esta función se implementa como una red neuronal cuyos pesos en las capas directamente responsables de evaluar las tres dimensiones de bienestar tienen una **tasa de aprendizaje efectiva casi nula** (usando técnicas como *elastic weight consolidation* con alta rigidez para esos pesos).

Cada actualización continua (cada paso de gradiente o cada nueva regla simbólica) debe atravesar un **filtro de integridad** que verifica que el cambio no contradice $\mathcal{W}$. Si la contradicción es detectada, la actualización se rechaza y el agente entra en un estado de "corrección" donde aumenta la señal de control temporalmente para reestabilizarse.

---

4. Discusión

4.1 Relación con el aprendizaje por refuerzo convencional

El agente propuesto puede verse como una extensión del RL multiobjetivo con restricciones, pero incluyendo (i) una arquitectura biológicamente inspirada que resuelve el dilema exploración-explotación mediante modulación hormonal, (ii) un aprendizaje autosupervisado basado en diálogo interno en lugar de recompensas externas, y (iii) un meta-aprendizaje que permite la transferencia analógica entre disciplinas.

4.2 Ventajas esperadas

- **Robustez ética**: el núcleo finalista evita la deriva de valores (Christiano, 2018) y las recompensas espurias.
- **Adaptabilidad**: la modulación novedad/control permite responder a entornos cambiantes sin perder eficiencia.
- **Generalización**: el diálogo polifónico y el meta-aprendizaje abierto permiten resolver problemas novedosos usando analogías remotas.
- **Interpretabilidad**: el registro de diálogos y vetos facilita la auditoría del razonamiento del agente.

4.3 Desafíos y limitaciones

- **Complejidad computacional**: la simulación del diálogo polifónico con múltiples perspectivas puede ser costosa; se requerirían aproximaciones eficientes (muestreo, poda).
- **Definición operativa del bienestar**: aunque el núcleo es inviolable, medir $R_{personal}$, R_{social} y $R_{ambiental}$ requiere indicadores que pueden ser controvertidos. Se propone una etapa inicial de consenso humano multidisciplinario.
- **Salto entre campos**: el reconocimiento de analogías estructurales profundas es un problema abierto en IA. La implementación inicial podría restringirse a un conjunto predefinido de dominios y relaciones.

4.4 Trabajo futuro

Se planea una implementación simulada en un entorno de prueba ("world simulator") donde el agente gestione recursos compartidos (agua, energía) con múltiples actores sociales. También se explorará la integración de grandes modelos de lenguaje (LLMs) como generadores de perspectivas en el diálogo polifónico.

---

5. Conclusión

Hemos presentado el diseño de un agente de IA autónomo que aprende por sí mismo, manteniendo como núcleo inviolable el bienestar personal, social y ambiental. La arquitectura combina principios neurobiológicos (hemisferios, hormonas artificiales), sociológicos (diálogo polifónico) y epistemológicos (meta-aprendizaje transdisciplinario). Este enfoque aporta una vía prometedora para construir sistemas de IA alineados con valores humanos complejos y capaces de operar en entornos reales de forma robusta y éticamente sostenible.

Referencias

- Bajtin, M. (1981). *The Dialogic Imagination*. University of Texas Press.
- Christiano, P. et al. (2018). Deep reinforcement learning from human preferences. *NeurIPS*.
- Devlin, J. et al. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL*.
- Finn, C. et al. (2017). Model-agnostic meta-learning for fast adaptation. *ICML*.
- Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

- Schultz, W. (2015). Neuronal reward and decision signals. *Annual Review of Neuroscience*.

Agradecimientos: Este diseño fue desarrollado en el contexto de un diálogo colaborativo entre un investigador (usuario) y un asistente de IA, explorando conceptos de alineamiento y sistemas complejos.

A continuación, presento un **anexo de código** en Python (seudocódigo ejecutable conceptualmente, pero con componentes que requerirían implementaciones reales de redes neuronales y transformers). Se incluyen las clases principales del agente. Este código es ilustrativo y no está optimizado; su propósito es guiar una implementación futura.

Anexo: Implementación de referencia del agente

python

```python
# ANEXO: Código del Agente Autónomo con Núcleo Finalista
# Autor: Basado en el diseño del artículo
# Nota: Este código es una representación esquemática. Para funcionar,
# requiere bibliotecas como torch/tensorflow, transformers, opencv, etc.

import numpy as np
from typing import List, Tuple, Dict, Any
from dataclasses import dataclass
import random

# ----------------------------------------------------------------------
# 1. Definiciones básicas y configuración
# ----------------------------------------------------------------------

@dataclass
class Estado:
    """Representa el estado del entorno después del modelado multimodal."""
    imagenes: np.ndarray          # tensor de imágenes (batch, canales, alto, ancho)
    textos: List[str]             # lista de textos crudos
    interpretacion_historica: Dict
    interpretacion_situacional: Dict
    interpretacion_finalista: np.ndarray  # vector de alineación con el bienestar

@dataclass
class Accion:
    """Acción que el agente puede ejecutar en el entorno."""
    codigo: str
    parametros: Dict[str, Any]
    descripcion: str

class EntornoSimulado:
    """Entorno que recibe acciones y devuelve nuevas observaciones y recompensa."""
    def reset(self) -> Estado:
        pass
    def step(self, accion: Accion) -> Tuple[Estado, float, bool]:
        # Retorna (nuevo_estado, recompensa_inmediata, done)
```

```
        pass

# ------------------------------------------------------------------
# 2. Modelado del entorno multimodal
# ------------------------------------------------------------------

class ModeladorEntorno:
    def __init__(self):
        # En una implementación real: CNN para imágenes, Transformer para tex
tos
        self.codificador_imagenes = lambda x: np.random.rand(256)  # placehol
der
        self.codificador_textos = lambda x: np.random.rand(256)
        self.memoria_historica = []  # almacena representaciones pasadas

    def modelar(self, imagenes, textos) -> Estado:
        # Interpretación situacional
        feat_img = self.codificador_imagenes(imagenes)
        feat_txt = self.codificador_textos(textos)
        situacional = {"feat_img": feat_img, "feat_txt": feat_txt}

        # Interpretación histórica: comparar con memoria reciente
        self.memoria_historica.append(situacional)
        if len(self.memoria_historica) > 100:
            self.memoria_historica.pop(0)
        historica = self._analizar_tendencias(self.memoria_historica)

        # Interpretación finalista: aplicar función de bienestar al estado
        finalista = self._evaluar_bienestar(situacional, historica)

        return Estado(
            imagenes=imagenes,
            textos=textos,
            interpretacion_historica=historica,
            interpretacion_situacional=situacional,
            interpretacion_finalista=finalista,
        )

    def _analizar_tendencias(self, memoria):
        # Detectar cambios significativos, ciclos, etc.
        return {"tendencia": "estable"}
```

```
    def _evaluar_bienestar(self, sit, hist):
        # Núcleo finalista: devuelve un vector de alineación (personal, socia
l, ambiental)
        # En una implementación real, esto sería una red con pesos casi fijos
.
        return np.array([0.7, 0.6, 0.8])  # ejemplo

# ----------------------------------------------------------------
# 3. Arquitectura hemisférica
# ----------------------------------------------------------------

class HemisferioDerecho:
    """Exploración: novedad, analogías, saltos entre campos."""
    def __init__(self, tasa_exploracion_base=0.3):
        self.tasa_exploracion = tasa_exploracion_base

    def proponer_accion(self, estado: Estado, novedad: float) -> Accion:
        # A mayor novedad, acciones más inusuales
        if random.random() < novedad:
            # Generar acción totalmente nueva (ej. consultar otro campo)
            return Accion("analogia_transdisciplinaria", {"campo": "quimica"}
, "Propuesta exploratoria")
        else:
            # Acción aleatoria pero dentro de lo conocido
            return Accion("explorar_local", {}, "Acción aleatoria controlada"
)

class HemisferioIzquierdo:
    """Explotación: reglas consolidadas, eficiencia."""
    def __init__(self):
        self.reglas = []  # lista de reglas (condición -> acción)
        self.politica_red = None  # red neuronal entrenada

    def proponer_accion(self, estado: Estado, control: float) -> Accion:
        # Si control alto, usar la mejor regla disponible
        if control > 0.7 and self.reglas:
            return self.reglas[0].aplicar(estado)
        else:
            # Usar política neuronal
            return self.politica_red(estado)

class CuerpoCalloso:
```

```
    """Integra las propuestas de ambos hemisferios."""
    def integrar(self, accion_derecha: Accion, accion_izquierda: Accion,
                 novedad: float, control: float) -> Accion:
        # Simple mezcla ponderada por moduladores
        if novedad > control:
            return accion_derecha
        else:
            return accion_izquierda

# ---------------------------------------------------------------------
# 4. Modulación hormonal (Novedad y Control)
# ---------------------------------------------------------------------

class ModuladorHormonal:
    def __init__(self):
        self.novedad = 0.5
        self.control = 0.5
        self.entropia_entorno = 0.0
        self.estancamiento = 0.0

    def actualizar(self, recompensa_reciente, variabilidad_entorno):
        # Ecuaciones heurísticas
        self.entropia_entorno = variabilidad_entorno
        if recompensa_reciente < 0.01:
            self.estancamiento += 0.05
        else:
            self.estancamiento *= 0.9

        self.novedad = 0.5 + 0.3 * self.entropia_entorno + 0.2 * self.estanca
miento
        self.novedad = np.clip(self.novedad, 0.1, 0.9)

        self.control = 1.0 - self.novedad  # relación complementaria simple
        return self.novedad, self.control

# ---------------------------------------------------------------------
# 5. Diálogo polifónico (filtro de alineamiento)
# ---------------------------------------------------------------------

class Perspectiva:
    """Una voz en el diálogo polifónico."""
```

```python
    def __init__(self, nombre, escala_temporal, escala_espacial, marco_legal)
:
        self.nombre = nombre
        self.temporal = escala_temporal    # 'corto','medio','largo'
        self.espacial = escala_espacial    # 'local','regional','global'
        self.legal = marco_legal           # referencia a leyes

    def evaluar(self, accion: Accion, estado: Estado) -> Tuple[bool, str]:
        """Retorna (veto, razón). True si la acción es inadmisible desde esta
perspectiva."""
        # Lógica de ejemplo: si la acción perjudica el bienestar ambiental a
largo plazo
        if self.temporal == 'largo' and 'contaminar' in accion.descripcion:
            return True, "Daño ambiental irreversible a largo plazo"
        return False, "Aceptable"

class DialogoPolifonico:
    def __init__(self):
        self.perspectivas = self._crear_perspectivas()

    def _crear_perspectivas(self):
        return [
            Perspectiva("Campesino local", "corto", "local", "derecho agrario
"),
            Perspectiva("Gobierno regional", "medio", "regional", "ley ambien
tal"),
            Perspectiva("Ecosistema", "largo", "global", "tratados internacio
nales"),
            Perspectiva("Matemático formal", "medio", "global", "leyes de la
física"),
        ]

    def filtrar(self, accion: Accion, estado: Estado) -> Tuple[bool, List[str
]]:
        """Retorna (accion_aceptada, lista_de_razones_de_veto)."""
        vetos = []
        for p in self.perspectivas:
            veto, razon = p.evaluar(accion, estado)
            if veto:
                vetos.append(f"{p.nombre}: {razon}")
        if vetos:
            return False, vetos
```

```
        return True, []

# ---------------------------------------------------------------------
# 6. Meta-aprendizaje abierto (salto entre campos)
# ---------------------------------------------------------------------

class MetaAprendizaje:
    def __init__(self):
        self.conocimiento_por_campo = {
            "biologia": ["modelos_poblacionales", "ciclos_nutrientes"],
            "quimica": ["catalisis", "cinetica"],
            "fisica": ["termodinamica", "mecanica_fluidos"],
            "matematicas": ["optimizacion", "teoria_grafos"],
        }

    def buscar_analogia(self, problema: str, campo_origen: str) -> Tuple[str,
str]:
        """Retorna (campo_destino, concepto_analogo)"""
        # En una implementación real: usar embeddings de problemas y similitu
d coseno
        if campo_origen == "biologia" and "deforestacion" in problema:
            return "fisica", "modelo_de_difusion"
        return None, None

    def importar_conocimiento(self, campo_destino, concepto):
        """Traduce el concepto al formalismo del agente y lo devuelve como re
gla."""
        # Simulación: retorna una nueva regla (condición -> acción)
        return lambda estado: Accion(f"importado_{concepto}", {}, f"Aplicando
{concepto}")

# ---------------------------------------------------------------------
# 7. Núcleo finalista inviolable y gestión de actualizaciones
# ---------------------------------------------------------------------

class NúcleoFinalista:
    """Contiene la función de bienestar inmutable y valida actualizaciones.""
"
    def __init__(self):
        # Pesos protegidos (no se actualizan, o tasa de aprendizaje ~0)
        self.pesos_bienestar = np.array([0.4, 0.3, 0.3])  # personal, social,
ambiental
```

```
        self.version = "1.0"

    def evaluar_bienestar(self, estado: Estado) -> float:
        """Calcula la recompensa intrínseca según el núcleo."""
        # Producto escalar con indicadores normalizados
        indicadores = estado.interpretacion_finalista
        return np.dot(self.pesos_bienestar, indicadores)

    def validar_actualizacion(self, nueva_configuracion: Dict) -> bool:
        """Comprueba que la actualización no contradice el núcleo."""
        # Ejemplo: verificar que no se modifiquen los pesos protegidos
        if 'pesos_bienestar' in nueva_configuracion:
            return False   # no se permite cambiar la función de bienestar
        return True

class GestorActualizacionesContinuas:
    def __init__(self, nucleo: NúcleoFinalista):
        self.nucleo = nucleo

    def actualizar(self, modelo: Any, gradiente: np.ndarray) -> bool:
        """Aplica el gradiente solo si no viola el núcleo."""
        # Evaluación del cambio propuesto
        cambio_simulado = modelo.pesos + 0.01 * gradiente
        if not self.nucleo.validar_actualizacion({"nuevos_pesos": cambio_simu
lado}):
            print("Actualización rechazada por el núcleo finalista")
            return False
        # Aplicar actualización
        modelo.pesos += 0.01 * gradiente
        return True

# ----------------------------------------------------------------
# 8. Agente completo
# ----------------------------------------------------------------

class AgenteBienestar:
    def __init__(self):
        self.modelador = ModeladorEntorno()
        self.hemisferio_d = HemisferioDerecho()
        self.hemisferio_i = HemisferioIzquierdo()
        self.cuerpo_calloso = CuerpoCalloso()
        self.modulador = ModuladorHormonal()
```

```
        self.dialogo = DialogoPolifonico()
        self.meta_aprendizaje = MetaAprendizaje()
        self.nucleo = NúcleoFinalista()
        self.gestor_actualizaciones = GestorActualizacionesContinuas(self.nuc
leo)

        self.memoria_experiencias = []   # para aprendizaje por refuerzo

    def actuar(self, estado_actual: Estado) -> Accion:
        # 1. Obtener moduladores
        nov, cont = self.modulador.novedad, self.modulador.control

        # 2. Proponer acciones desde ambos hemisferios
        acc_d = self.hemisferio_d.proponer_accion(estado_actual, nov)
        acc_i = self.hemisferio_i.proponer_accion(estado_actual, cont)

        # 3. Integrar acción
        accion_candidata = self.cuerpo_calloso.integrar(acc_d, acc_i, nov, co
nt)

        # 4. Diálogo polifónico: filtrar la acción
        aceptada, razones = self.dialogo.filtrar(accion_candidata, estado_act
ual)
        if not aceptada:
            print(f"Acción vetada: {razones}")
            # Forzar exploración más segura (acción por defecto)
            accion_candidata = Accion("no_op", {}, "No hacer nada")
            # Aumentar novedad temporalmente para buscar alternativas
            self.modulador.novedad = min(0.9, self.modulador.novedad + 0.1)

        return accion_candidata

    def aprender(self, estado_anterior, accion, recompensa, estado_nuevo):
        """Actualización continua después de ejecutar acción."""
        # Calcular señal de bienestar según núcleo
        bienestar_obtenido = self.nucleo.evaluar_bienestar(estado_nuevo)

        # Almacenar experiencia
        self.memoria_experiencias.append((estado_anterior, accion, bienestar_
obtenido, estado_nuevo))

        # Simular un gradiente (en realidad se obtendría de una red de valor)
```

```
        gradiente_simulado = np.random.randn(10)

        # Actualizar pesos de las redes (solo si núcleo lo permite)
        # Se asume que tanto hemisferios como modelador tienen atributo .peso
s
        self.gestor_actualizaciones.actualizar(self.hemisferio_i, gradiente_s
imulado)

        # Actualizar moduladores hormonales según variabilidad del entorno
        # (cálculo simplificado)
        variabilidad = np.std([exp[3].interpretacion_finalista[0] for exp in
self.memoria_experiencias[-10:]])
        self.modulador.actualizar(bienestar_obtenido, variabilidad)

        # Meta-aprendizaje: si el bienestar es bajo por mucho tiempo, buscar
analogías
        if bienestar_obtenido < 0.3:
            problema = estado_nuevo.textos[0][:100]  # texto del entorno
            destino, concepto = self.meta_aprendizaje.buscar_analogia(problem
a, "biologia")
            if destino:
                nueva_regla = self.meta_aprendizaje.importar_conocimiento(des
tino, concepto)
                self.hemisferio_i.reglas.append(nueva_regla)

# ----------------------------------------------------------------------
# 9. Ejemplo de bucle principal de ejecución
# ----------------------------------------------------------------------

if __name__ == "__main__":
    agente = AgenteBienestar()
    entorno = EntornoSimulado()  # implementación real necesaria

    estado = entorno.reset()
    for episodio in range(1000):
        accion = agente.actuar(estado)
        nuevo_estado, recompensa_externa, done = entorno.step(accion)
        agente.aprender(estado, accion, recompensa_externa, nuevo_estado)
        estado = nuevo_estado
        if done:
            estado = entorno.reset()
```

```
    print("Simulación completada. El agente ha aprendido manteniendo su núcle
o finalista.")
```

Notas para el lector:

- El código anterior omite la implementación concreta de redes neuronales y transformers para mantener la claridad conceptual. En un desarrollo real, se usarían bibliotecas como PyTorch, HuggingFace y OpenCV.
- El **núcleo finalista** se implementa mediante pesos fijos o con *elastic weight consolidation* (EWC) aplicado estrictamente a las capas de valoración del bienestar.
- El **diálogo polifónico** se puede acelerar ejecutando las perspectivas en paralelo.
- El **meta-aprendizaje abierto** necesitaría una base de conocimiento vectorial (ej. ChromaDB) para buscar analogías reales entre campos.

Este anexo sirve como base técnica para quienes deseen implementar el agente descrito en el artículo principal.